International Conference on Inter Disciplinary Research in Engineering and Technology [ICIDRET]

ISBN	978-81-929742-5-5
Website	www.icidret.in
Received	14 - February - 2015
Article ID	ICIDRET037

Vol	I
eMail	icidret@asdf.res.in
Accepted	25 - March - 2015
eAID	ICIDRET.2015.037

Performance Evaluation of Alternative file systems over HDFS

Udhayakumar Shanmugam[1], Dhinakaran K[2], Silviyanancy J[3], Nikhiljacob[4]
[1]Research Scholar, [2]Assistant Professor, Assistant Professor,[4]Student
Department of computer Science and Engineering,
[1]B.S.Abdur Rahman University, [2]Saveetha University, [3]Saveetha University.

Abstract-*In the last decade or so there has been an increased use and growth of social media, unconventional web technologies, computers and mobile applications, which have all encouraged development of various database models. Recent datasets are extremely costly and unpractical to administer with SQL databases due to lack of structure, high scalability, and elasticity that is needed. The No SQL data stores such as Mongo DB and Cassandra provide a desirable platform for fast and efficient data queries. With the introduction of the "Big Data" the size and structure of data have become highly dynamic and complex. This paper exhibits evaluation of the Cassandra a No SQL database when used in conjunction with the Hadoop Map Reduce engine, also the Ceph File system developed by Data Stax and the Lustre File system which all can be used as an alternative to HDFS (Hadoop Distributed File System). We provide a brief overview of MapReduce and Hadoop and then show some of the shortcomings of Hadoop + HDFS. Ceph maximizes the separation between data and metadata management by replacing allocation tables with a pseudo-random data distribution function. Also we have evaluated theoretical and actual performance of Lustre and HDFS for a variety of workloads in both traditional and Map/Reduce-based applications.*

Keywords: Hadoop, Bigdata, Ceph file system, HDFS. Cassandra, luster file system

I INTRODUCTION

With the outbreak of "Big Data" the complexity, size and nature of data has become unpredictable and unimaginable. In one day on a social media millions of data are getting uploaded as well as used. Since the data is constantly being modified through social media, newsfeeds, and scientific sensor input, then the requirements for the storage models also have the necessity changes. However the MapReduce model has evolved as the paradigm of choice for "Big Data" processing. The Map/Reduce is a distributed computational algorithm designed by Google which is used for a wide variety of large-scale jobs. The Hadoop Distributed File System (HDFS) is one which defines HDFS as the primary storage system used by Hadoop applications. HDFS creates multiple replicas of data blocks and distributes them on compute nodes throughout a cluster to enable reliable and extremely rapid computations. Hadoop utilizes a scale-out architecture that is configured as a cluster. As the process starts, it cites the states and data in Hadoop it is broken down into blocks and spread throughout a cluster. As soon as it happens the MapReduce tasks can be carried out on the smaller subsets of data that may make up a very large dataset overall which accomplishes the type of scalability needed for big data processing.

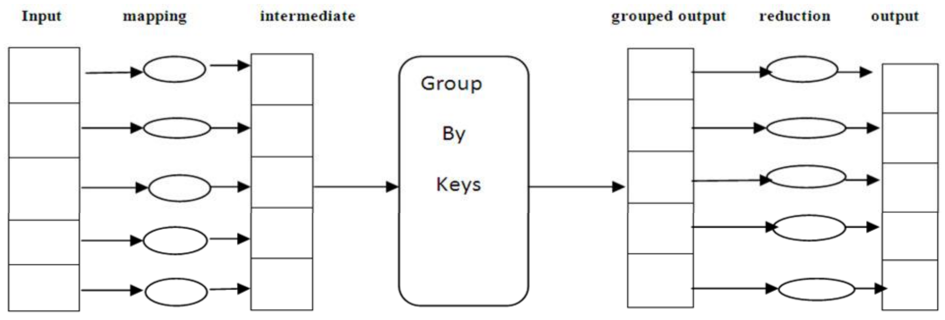
The studies with performance in mind and the applicability of the MapReduce model to No SQL DB, such as Mongo DB [16] and Cassandra, have been lacking. Since the data now-a-days increasingly produced from various sources are increasingly unstructured while they continually growing in size with user interaction and so it is important to evaluate the No SQL model when used with the MapReduce processing algorithm. In this paper, we analyse the performance when using Cassandra as the data store and Apache Hadoop for processing [17]. Cassandra is an open source non-relational, column oriented distributed database for storing large amounts of unstructured data.

The centralization that is inherent in the client/server model has proved to be a significant obstacle to scalable performance. Recently distributed file systems have adopted architectures based on object-based storage, in which conventional hard disks are replaced with intelligent object storage devices (OSD). It replaces the traditional block-level interface with one in which clients can read or write byte ranges to much larger named objects, distributing low-level block allocation decisions to the devices themselves. Clients mostly interact with a metadata server (MDS) to perform metadata operations while communicating directly with OSDs to perform file significantly improving overall scalability. Ceph is a distributed file system that provides excellent performance and reliability [1]. The architecture is based on that systems at the petabyte scale are inherently dynamically. It decouples the data and metadata operations by eliminating file allocation tables and replacing them with generating functions. This allows them to leverage the intelligence present in OSDs to distribute the complexity surrounding data access, update serialization, replication and reliability, failure detection, and recovery. Ceph file system makes use of highly adaptive distributed metadata cluster architecture that dramatically improves the scalability of metadata access. Lustre is a client/server based cluster file system where data are stored on Object Storage Servers (OSSs) and metadata are stored on Metadata Servers (MDSs). It is designed for large-scale computing. Lustre is optimized to operate efficiently on many types of high-end network fabrics by taking advantage of RDMA where available. In Lustre files are broken into stripes, which are typically stored on multiple Object Storage Targets (OSTs), allowing parallel read and writes access to different parts of the file. Lustre is POSIX-compliant and mounted remotely similar to NFS [2]

II BACKGROUND UNDERSTANDING

1. Map Reduce and HDFS

The Map Reduce algorithm defines splitting a data set to process in parallel over a cluster. Whereas inputs, scheduling, parallelization and machine failures are handled by the framework itself and monitored by a node called the master [5]. It splits the parallel execution into two phases: map and reduce where the Map processes a key and produces a set of intermediate key/value pairs. The reduce phase uses the intermediate results to construct the final output. Hadoop is the most popular open source implementation of the model. It consists of two core components namely The Hadoop Map Reduce Framework and the Hadoop Distributed File System [6]. Hadoop MapReduce consists of a Job Tracker that runs on the master node and Task Trackers running on each of machine. Job Tracker is responsible for determining job specifications, submitting the user job to the cluster and monitoring workers and the job status [7]. Task Trackers execute the user specified map or reduce tasks. It relies on HDFS for data distribution and input management which automatically breaks data into chunks and spreads them on the cluster where the nodes hosting the input splits and replicas are called Data Nodes [8]. Each and every Task Tacker processes the input chunk hosted by the local Data Node that is done to leverage data locality. The input splits are replicated among the Data Nodes based on a user set replication-factor. This design prevents data loss and helps with fault tolerance in case of node failures.



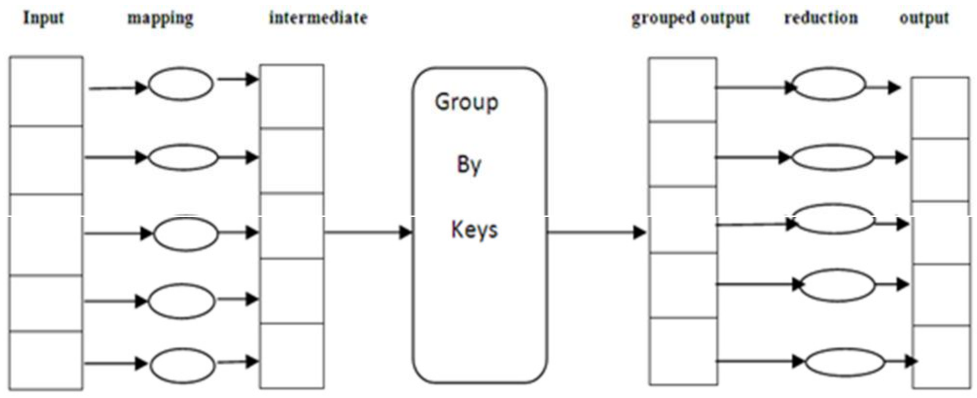


Figure 1: Parallel Execution of Map Reduce

2. Cassandra

Cassandra was developed by Facebook which is an open source, non-relational, column oriented, distributed database, developed for storing large amounts of unstructured data over commodity servers. It is a peer-supermodel which makes it not only tolerant against single points of failure but also easily horizontally scalable. A cluster in Cassandra can be expanded on demand by simply starting new servers which only know the address of the node to contact for getting the start-up information [17].The column is the lowest/smallest increment of data. The tuple contains a name, a value and a timestamp. The interface definition of a Column:

```
struct Col {
1: bin name,
2: bin value,
3: i64 time_stamp,
}
```

2.1 Data Model

Figure 2 shows the column oriented data model of Cassandra. Here a column is the smallest component of data and it is a tuple of name, value and time stamp. They are used for conflict resolution as 1ultiple versions of the same record may be present. The Columns associated with a certain key can be depicted as a row which does not have a pre-determined structure as each of them may contain several columns. A column family is a collection of rows similar to a table in a relational database. Here the Column families are stored in separate files which are then sorted by row key order. The placement of rows on the nodes of a Cassandra cluster depends on the row key and the partitioning strategy. The Key spaces are containers for column families just as databases have tables in RDBMSs

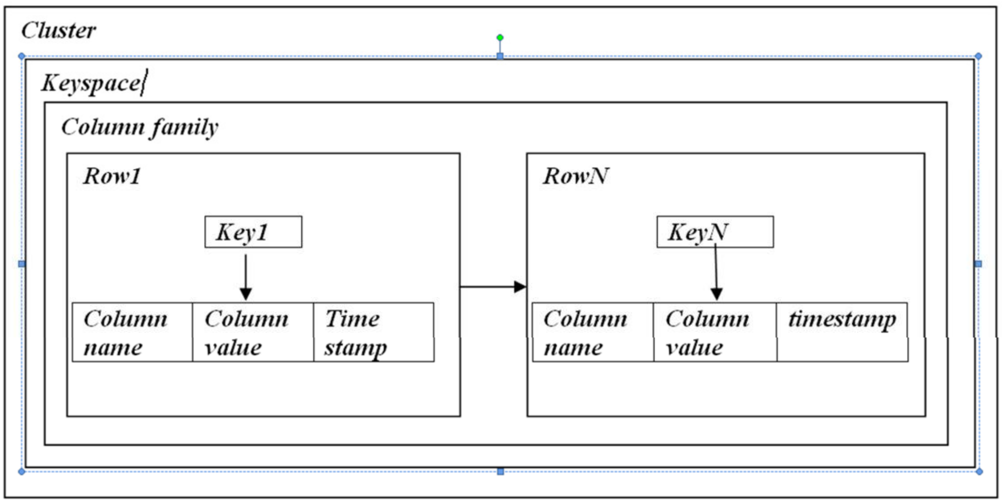


Figure 2:Heirarchy of Cassandra Model

Representation of the row key -> column families ->column structure is

```
{
"mcv":
{"allusers":
{"eAddress"
{"name":"eAddress", "value":"koo@bar.com"},
"webSite":
{"name":"web","value":"http://bar.com"}
},
"Stat":
{
"visits":{"name":"visits","value":"123"}
}
},
"u2":{
"allusers":{
"elAddress":
{"name":"eAddress","value":"u2@bar.com"},
"twitter":{"name":"twitter","value":"u2"}
}
}
}
```

Now the key "mcv" identifies data in two different column families namely "all users" and "Stat". Thus does not show that data from these column families is related. The semantics of having data for the same key in two different column families is entirely up to the application. Also note that within the "Users" column family, "mcv" and "us2"have different column names defined. This is perfectly valid in Cassandra

2.2 Reliability and consistency

Cassandra has automatic replication to duplicate records throughout the cluster by a user set replication-factor which ensures that failing nodes do not result in data loss. Cassandra also offers configurable consistency which provides the flexibility to consciously make trade-offs between latency and consistency.

2.3 Partitioning of data

Cassandra provides two main partitioning strategies namely RandomPartitioner and ByteOrderedPartitioner. The RandomPartitionr is the default strategy and it is used in most cases which makes use of consistent hashing to evenly distribute rows across the cluster. The hashing algorithm is used to create an md5 hash value of row key. All Cassandra node has a token value that specifies the range of keys for which they are responsible. Based on the hash and range tokens a row is stationed in the cluster. Distributing the records evenly throughout the cluster balances the load by spreading out client requests.

2.4 Read and Write Operation

The client can contact any Cassandra node for any operation. The nodes are connected to each other to serves as a coordinator. Thus the coordinator forwards the client request to the replica nodes owning the data being claimed. For all write request first a commit log entry is created then mutated columns are written to an in-memory structure called Memtable where upon reaching its size limit is committed to disk as a new SS Table. Each of this operation is executed as a background process. A write request is sent to all replica nodes but the consistency levelis determined by how many of them have to wait for a write transaction to be considered completed. When a read request is given the coordinator contacts the replica nodes specified by the consistency level

2.5 Support of Hadoop:

Cassandra and Hadoop integration is important as it provides data management and real-time analysis along with complex data intensive processing. The Hadoop-Cassandra cluster the Cassandra servers are overlapped with Hadoop TaskTrackers and DataNodes to ensure data locality so that each TaskTracker processes the data that is stored in the local Cassandra node. The DataNodes are required because Hadoop needs HDFS for copying the dependency jars, static and intermediary data [18].

3. Ceph:

The Ceph file system has three main components namely the client, a cluster of OSDs and a metadata server cluster the client at each instance of which exposes a near-POSIX file system interface to a host or process. A cluster of OSDs which collectively stores all data and metadataserver cluster manages the namespace while coordinating security, consistency and coherence of applications and to improve system performance [3, 4].The primary goals of the architecture are scalability performance and reliability. Our target workload may include such extreme cases as tens or hundreds of thousands of hosts concurrently reading from or writing to the same file or creating files in the same directory. In such situations the distributed file system workloads are inherently dynamic with significant variation in data and metadata access as active applications and data sets changeover time. Ceph addresses this issue of scalability by simultaneously achieving high performance, reliability and availability via three fundamental design features namely decoupled data and metadata, dynamic distributed metadata management, and reliable autonomic distributed object storage [9].

3.1 Data and Metadata Decoupled

Ceph increases the separation of file metadata management from the storage of file data. Metadata operations are collectively managed by a metadata server cluster while clients interact directly with OSDs to perform file I/O operations. Object-based storage has long promised to improve the scalability of file systems by delegating low-level block allocation decisions to individual devices. But object based file systems replace long per-file blacklists with shorter object lists which eliminate allocation lists entirely and the file data is striped onto predictably named objects through a special-purpose data distribution function called CRUSH which assigns objects to storage devices.

3.2 Distributed Metadata Management Dynamically

Since file system metadata operations make up as much as half of typical file system workloads effective metadata management is critical to overall system performance. Ceph utilizes a novel metadata cluster architecture based on Dynamic Sub tree Partitioning that distributes responsibility for managing the file system directory hierarchy among tens or even hundreds of MDSs. A specific partition preserves locality in each MDS's workload providing efficient updates and aggressive prefetching to improve performance for common workloads allowing Ceph to effectively utilize available MDS resources under any workload and achieve near linear scaling in the number of MDSs.

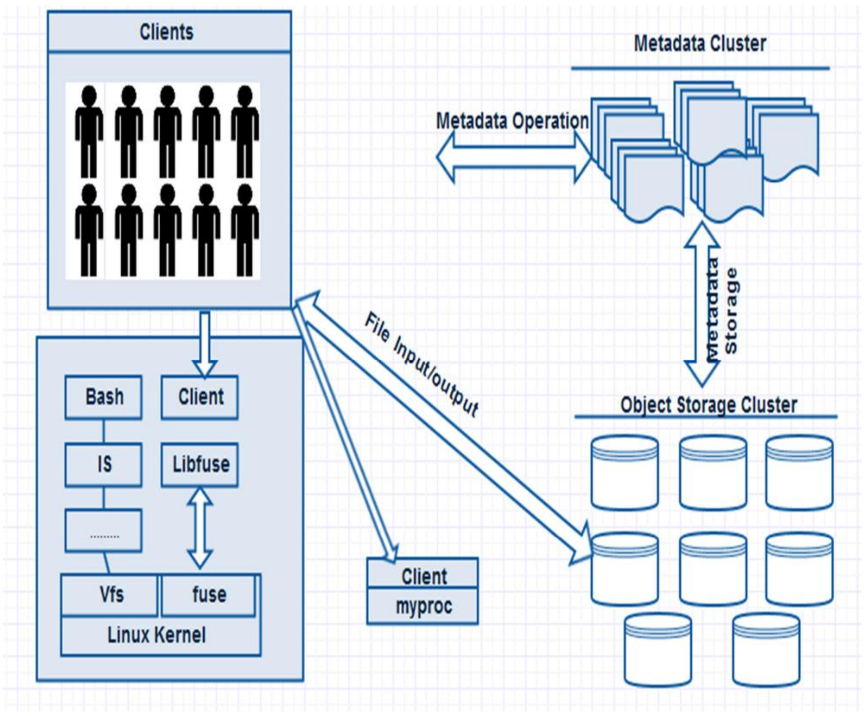


Figure 3: System Architecture

3.3 Reliable Distributed Object Storage

Many large systems are composed of many thousands of devices that are inherently dynamic. Ceph delegates responsibility for data migration, replication, failure detection and failure recovery to the cluster of OSDs that store the data. The OSDs collectively provide a single logical object store to clients and metadata servers.

4. Lustre

The use of clustered file systems as a backend for Hadoop storage improves performance of distributed file systems such as Lustre, Ceph, PVFS and GPFS with Hadoop has been compared to that of HDFS. With various optimizations and tuning efforts, a clustered file system can reach parity with HDFS. However, a consistent limitation in the studies of HDFS and non-HDFS performance with Hadoop is that they used the network infrastructure to which Hadoop is limited. In HPC environments where much faster network interconnects are available significantly better clustered file system performance with Hadoop is possible.

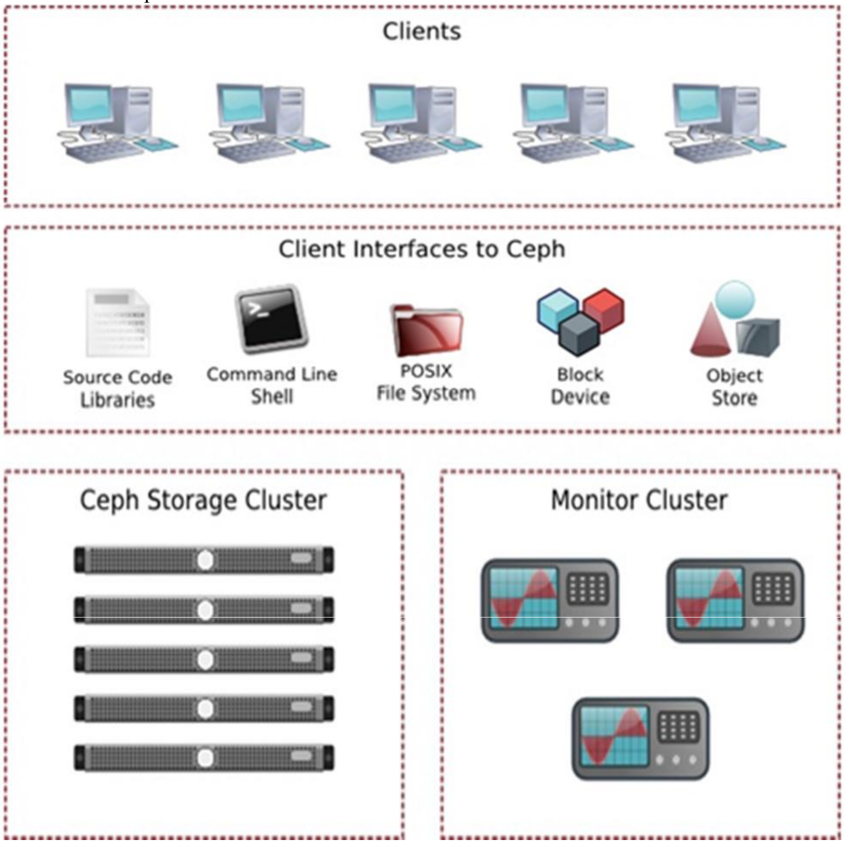
III PERFORMANCE EVALUVATION

A. Efficiencies of alternative file systems:

1. Cassandra: The objective is to offer easy Hadoop integration for Cassandra users. In HDFS Name Node service, that tracks each files metadata and block locations, is replaced with the "I node" columnfamily. Web applications that rely on fast data access NoSQL key-value store. Hadoop has fast access to data streaming into Cassandra from web users. T hemain design goals for the Cassandra File System were to first, simplify the operational overhead of Hadoop by removing the single points of failure int he HadoopNameNode.

2. Ceph:

They uniquely deliver object, block, and file storage in one unified system. Its features are a high- performance parallel file system that makes it a candidate to replace hadoop. Ceph asa "fully open source, distributed object store, network block device, and POSIX-compatible distributed file system designed for reliability, performance, and scalability." It's uniqueness comes in part because Ceph does all these things within a unified platform.

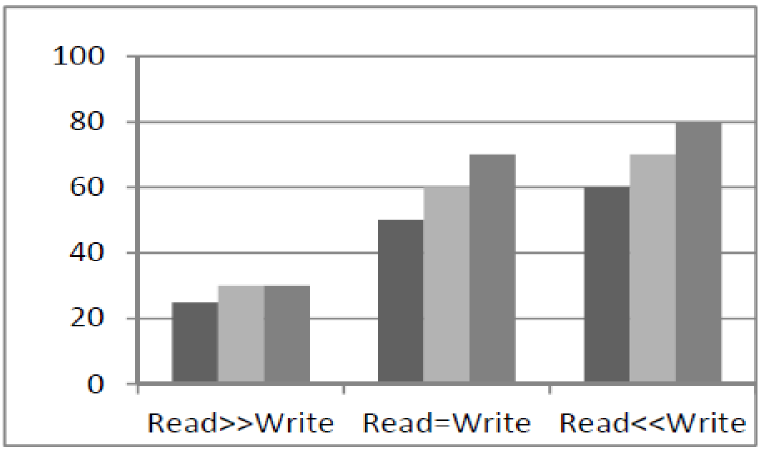


B. Conjunction of Hadoop with Alternative systems:

1. Cassandra with HDFS: With the conjunction of MapReduce and Cassandra the Hadoop was used with 3 different configurations:

- Hadoop-native, HDFS for input and output placement.
- Hadoop-Cassandra-FS, reads the input from Cassandra and writes the output to a FS shared by the workers.
- Hadoop-Cassandra-Cassandra reads input from Cassandra and writes output back to Cassandra.

Figure 4 shows the performance for 3 different Hadoopsetups under three different workloads which are classified in three different cases based on the input size to output size ratio. Figure 4a shows the case where the input dataset is significantly larger than the output. Like altering satellite image data by removing undesired areas to create high value images. The images are collected in a Cassandra cluster to provide search and query capabilities. In Figure4a we show that while HadoopCassandra-Cassandra is 1.1times slower than Hadoop-native at processing 4 million input records it gets only 1.8 times slower for 64 million. When increase the input 16 times does not lead to as big a performance difference.



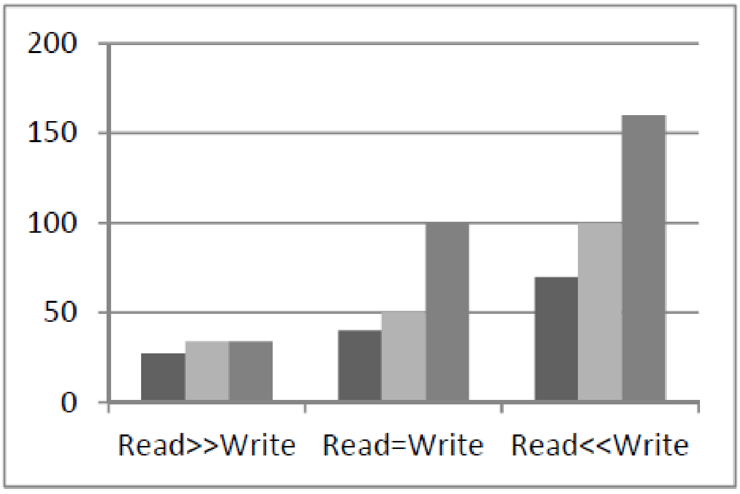


Figure 4: Hadoop with and without Cassandra under three different workloads

Hadoop-Cassandra-Cassandra and Hadoop-Cassandra-FS display very similar output where the write location does not have a considerable effect on performance. Therefore, each of the setups completes output writing in under 1.2seconds. But Hadoop reads 4 million records from HDFS1.3 times faster than it does from Cassandra; it is only 2.1times faster for 64 million. Thus shows that increasing the read size dramatically does not affect Hadoop-Cassandra performance to a great deal which is desirable for processing expeditiously dilating datasets. In Figure 4b,we use a workload where the output size is very close to, if not the same as, the input size thus Figure 4b shows that Hadoop-native is 1.2 to 1.4 times faster than Hadoop-Cassandra-FS and 1.9 to 2.9 times from Hadoop-Cassandra-Cassandra[17].

In the MapReduce computation is moved to the data which means each worker node processes the data split they own. Here data is split and distributed evenly among the Data Nodes and each Task Tracker processes the data from the local Data Node. By using Cassandra with Hadoopleaves the distribution of data to the partitioning strategy set for the input and output column families within Cassandra. The Random Partitioner will evenly split thedata among the nodes. This means that with RandomPartitioner each Hadoop worker has local access toalmost the same amount of data. Whereas the ByteOrderedPartitioner causes the data splits to be collected on a small set of nodes.WithHadoop-Cassandra-Cassandra using RandomPartitioner versus Byte Ordered Partitioner. It shows that using RandomPartitioner is 1.2 times faster for 4 million records while it is 3 times for 32 million. Even the performance also hangs dramatically with growth of the the data size as more data movement is required to complete the MapReduce job. Efficient MapReduce performance, it is crucial for the underlying data storage to distribute data evenly and allow for data locality. Distributed storage systems like HDFS store data in multiple nodes to avoid data loss in case of node failures.The data is replicated by user set replication factor andeach replica is placed on a different node. By using Cassandra with Hadoop, HDFS does not have any control of data replication. Therefore, if the data is not replicated through Cassandra itself node failures would result in data.

Figure 5 shows the performance of CPU and memory intensive jobs with Hadoop and Cassandra. Apart from it data intensive, the application memory and CPU demands are also shown to affect performance in various MapReduce implementations [18]. In most MapReduce applications it is observed that the output is significantly smaller than the input, since the output being small makes the write time negligible and consequently Hadoop-Cassandra-Cassandra and HadoopCassandra-FS display similar times. In Figure 5(a) HadoopCassandra-Cassandraunder CPU intensive workload shows closer times to Hadoop-native for each data point shown hereHadoopnativeis only up to 1.1 times faster.

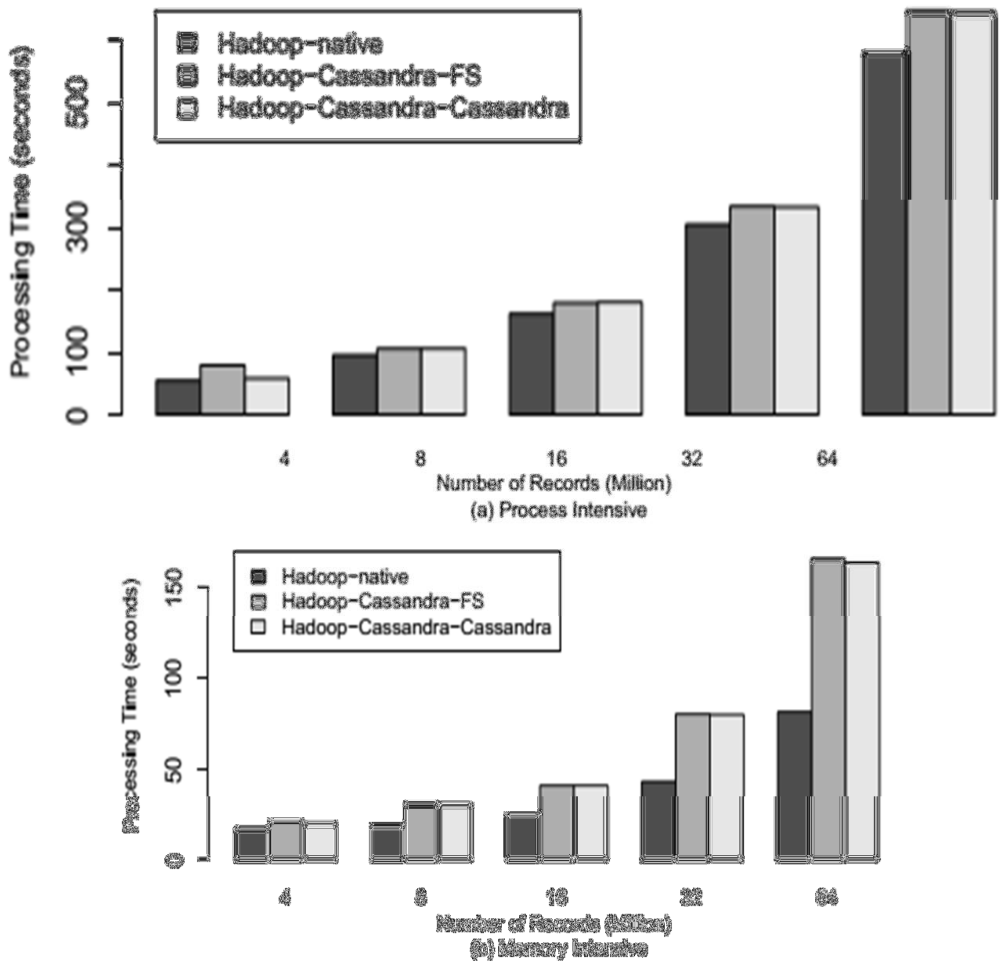


Figure 5: Running Hadoop with different setups for processing intensive operation

Thus shows tha the processing of map and reduce operations takes up most of the total job time and input/output locations changes only a little. In Figure 5(b)Hadoop-native performs 1.7 times faster than Hadoop-Cassandra-Cassandra for 4 million input records and thisratio reaches 2 as data increases to 64 million. In a Hadoop-Cassandra-Cassandra setup each worker node runs a Cassandra server in addition to a TaskTracker and DataNode. Cassandra servers have their own memory load on the worker machines because there are in-memory structures like Me tables and other cached data .Only for4 million input records Hadoop native is 1.7, and for 64million it is 2 times faster than Hadoop-Cassandra-Cassandra. Cassandra memory footprint is more affected by the number of column families than the data size in one of them and contributes to the overall slower performance[18].

A. Lustre in on juntion with HDFS:

According to the tasks-assigned strategy Hadoop cannot make task data local. For example when node1 (rack 2)requests a task, but all tasks pre-assign to this node has finished, then JobTracker will give node1 a task preassigned to other nodes in rack 2. In this situation, node1will run a few tasks whose data is not local. This breaks the Hadoop principle that Moving Computation is Cheaper than Moving Data. This generates a huge net I/O when these kinds of tasks have huge inputs.Hadoop+HDFS storage strategy of temporary or intermediate data is not good for high computational complexity applications which generate big and increasing MapTask outputs. Reduce node need to use HTTP to shuffle all related big MapTask outputs before real task begins, which will generate lots of net I/O and merge/spilloperation and also take up mass resources. Even worse, thebursting shuffle stage will make memory exhausted and make kernel kill some key threads according to java's wasteful memory usage, this will make the cluster unbalance and instable. HDFS cannot be used as a normal file system, which makes it difficult to extend. HDFS is time consuming for small files.

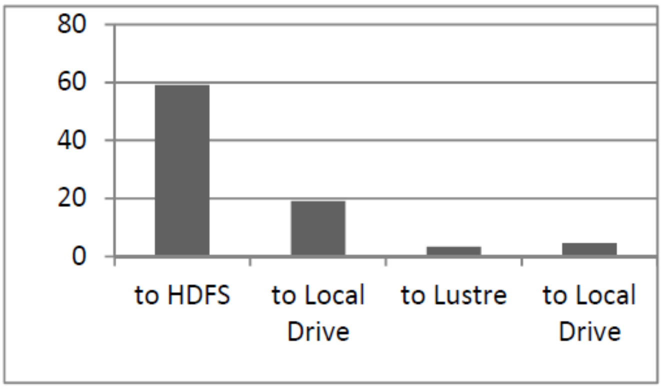


Figure 6: time to transfer 1gb file from local disk

We compare the differences of HDFS and Lustre in each stage. Map input: read/write

1. With block's location information
   a. HDFS: streaming read in each task, most locally, rare remotely network I/O.
   b. Lustre: read in parallel in each task from Lustre client.
2. Without block's location information.
   a. HDFS: streaming read in each task, most locally, rare remotely network I/O.
   b. Lustre: reads in parallel each task from Lustre client, less network I/O than the first one because of location information. Map output: read/write.
   c. HDFS: writes on local Linux file system, not HDFS.
   d. Lustre: writes on Lustre

A record emitted from a map will be serialized into a buffer and metadata will be stored into accounting buffers. When either the serialization buffer or the metadata exceed a threshold, the contents of the buffers will be sorted and written to disk in the background while the map continues to output records. If either buffer fills completely while the spill is in progress the map thread will block

IV CONCLUSION

In this paper we have paired Cassandra with Apache Hadoop and showed the performance gains and pitfalls of using the two together. Also we have determined what NoSQL platform might be suitable for Hadoop coupling where first we show a range of features conducive to efficient performance with the MapReduce model. Increasing the replication-factor on Cassandra does not affect Hadoopturn around time leveraging range scans reduces read repair calls on replicas; immunizing Hadoopfrom replication related performance degradation. Replacing HDFS with Lustre as the underlying file system for Hadoop compute clusters in HPC environments can the oretically result in significant improvements in system performance and cluster efficiency or potentially offer a lower overall system cost. Relative performance is expected to vary significantly based on a particular task's input/output pattern; we highly recommend prototyping the performance for a cluster's anticipated tasks before selecting a backend file system. Ceph addresses three critical challenges of storage systems scalability, performance, and reliability by occupying a unique point in the design space.

REFERENCE

[1] Adya, W. J. Bolosky, M. Castro, R. Chaiken, G.Cermak, J. R. Douceur, J. Howell, J. R. Lorch,M.Theimerand R. Wattenhofer. FARSITE: Federated,available, and reliable storage for an incompletelytrusted environment. In Proceedings of the 5thSymposium on Operating Systems Design andImplementation (OSDI), Boston, MA,Dec. 2002. USENIX
[2] P. J. Braam. The Lustre storage architecture.http://www.lustre.org/documentation.html, ClusterFileSystems, Inc., Aug. 2004.
[3] P. F. Corbett and D. G. Feitelson. The Vesta parallelfile system. ACM Transactions on ComputerSystems,14(3):225–264, 1996.
[4] S. Ghemawat, H. Gobioff, and S.-T.Leung. TheGooglefile system. In Proceedings of the 19th ACMSymposiumon Operating Systems Principles (SOSP'03), BoltonLanding, NY, Oct. 2003. ACM
[5] MapReduce:http://labs.google.com/papers/mapreduce-osdi04.pdf
[6] Hadoop:http://Hadoop.apache.org/core/docs/r0.20.0/mapred_tutorial.html
[7] JobTracker:http://wiki.apache.org/Hadoop/JobTracker
[8] TaskTracker:http://wiki.apache.org/Hadoop/TaskTracker
[9] Sage A. Weil, Scott A. Brandt, Ethan L. Miller,Darrell D. E. Long, Carlos Maltzahn, "Ceph: AScalable, High-Performance Distributed FileSystem",University of California, Santa Cruz

[10] "Comparing the Hadoop Distributed File System(HDFS) with the Cassandra File System (CFS)",BYDATASTAX CORPORATION, August 2013
[11] Amazondynamodb:http://aws.amazon.com/dynamodb/
[12] Apache Hadoop. http://hadoop.apache.org
[13] Apache HBase. http://hbase.apache.org
[14] Datastax. http://www.datastax.com/
[15] Datastaxenterprise.http://www.datastax.com/products/enterprise
[16] Mongodb. http://www.mongodb.org
[17] F. Cooper, A. Silberstein, E. Tam, R.Ramakrishnan, and R. Sears.Benchmarking cloudserving systems with ycsb. In Proceedings of the1stACM symposium on Cloud computing, SoCC '10,pages 143–154,New York, NY, USA, 2010. ACM
[18] ElifDede, BedriSendir, Pinar Kuzlu, Jessica Hartog,MadhusudhanGovindaraju,"An Evaluation ofCassandra for Hadoop",Grid and Cloud ComputingResearch LaboratorySUNY Binghamton, New York,USA
[19] sage weil,"ceph day ", november 2, 2012